LISTS & DICTIONARIES

LECTURE 05–2

JIM FIX, REED COLLEGE CSCI 121

MORE ON PROJECT 1: GRID

- **REMINDER:** A checkpoint for Project 1 is due **Thursday.**
  - ➡ Added a Gradescope problem to submit `rules.py` and `demo.py`
  - ➡ Just want to see that you've completed ***three rules***.

- **NOTE:** I just updated the Project 1 page.
  - ➡ There is an enhanced `Grid.py` available under "Set Up"...
    - – Linked as `project1-with-save.zip`.
  - ➡ These enhancements are described under "Update"...
    - – Can help you build a richer demo next week.

- **[[[DEMO of my SOLUTION and also some COOL RULES]]]**

READING FOR PYTHON LISTS

- **Reading:**
  - TP Ch 8-10
  - CP Ch 2.1-2.4

OUR FIRST DATA STRUCTURE: PYTHON LISTS

- Python lets you represent sequences of data values:

```
>>> xs = [2,3,7,15,100]
>>> xs
[2, 3, 7, 15, 100]
>>> xs[3]
15
>>> xs[0]
2
>>> len(xs)
5
```

- This is a built-in data structure called a Python ***list***.
  - A list is a *sequence* of numbered slots; each slot stores a value.
  - Each slot can be accessed by its ***index***, starting at 0.
  - A list has a ***length***.
- A Python list is also our first explicit example of a Python (data) ***object***

MODIFYING A LIST'S CONTENTS

- A Python list is a ***mutable*** data structure.
  - ➡ This means that its contents can be changed.

```
>>> xs
[2, 3, 7, 15, 100]
>>> xs[3]
15
>>> xs[3] = 200
>>> xs[3]
200
>>> xs
[2, 3, 7, 200, 100]
>>> xs[0] = xs[2] + xs[4]
>>> xs
[107, 3, 7, 200, 100]
>>> xs[4] = 1000
>>> xs
[107, 3, 7, 200, 1000]
```

LIST INDEXING

- You have to be careful when accessing a list; need to be mindful of its length.

```
>>> xs = [2,3,7,15,100]
>>> xs
[2, 3, 7, 15, 100]
>>> xs[5]
error!
```

- Using a negative index allows you to access backward from the end of the list:

```
>>> xs[-1]
100
>>> xs[-2]
15
>>> xs[-5]
2
>>> xs[-6]
error!
```

EXAMPLE LIST FUNCTION

- This checks a list to see if its contents read the same backwards as forwards:

```
def is_palindrome(xs):
    hi = len(xs)-1
    lo = 0
    while hi > lo:
        if xs[lo] != xs[hi]:
            return False
        lo = lo + 1
        hi = hi - 1
    return True
```

EXAMPLE LIST FUNCTION

- This does the same using negative indexing

```
def is_palindrome(xs):
    index = 0
    middle = len(xs) // 2
    while index < middle
        if xs[index] != xs[-(index+1)]:
            return False
        index = index + 1
    return True
```

EXAMPLE LIST FUNCTION

- This checks to see if the contents of two lists are the same:

```
def same_contents(xs,ys):
    if len(xs) != len(ys):
        return False
    i = 0
    while i < len(xs):
        if xs[i] != ys[i]:
            return False
        i = i + 1
    return True
```

EXAMPLE LIST FUNCTION

- This checks to see if the value **`y`** is stored in any of the slots of the list **`xs`**:

```
def contains(y,xs):
    i = 0
    while i < len(xs):
        if xs[i] == y:
            return True
        i = i + 1
    return False
```

LIST CONTENT CHECKS

- Python has `contains` and `same_contents` built into its language:

```
>>> 4 in [1,2,4,8] # Does the list contain an element?
True
>>> 7 in [1,2,4,8]
False
>>> xs = [1,3,4]
>>> ys = [1,3,5]
>>> xs == ys          # Are the lists' contents the same?
False
>>> xs != ys
True
>>> ys[2] = 4
>>> xs == ys
True
>>> xs != ys
False
>>> xs is ys          # Are they the same list object?
False
```

- The operators `in` and `==` check *contents*. The operator `is` checks list ***identity***.

MODIFYING LISTS: ADDING AND INSERTING

- We can add more slots to a list object:

```
>>> xs = [13,5,71]
>>> xs
[13, 5, 71]
>>> xs.append(-57)          # Adds a new slot to the end.
>>> xs
[13, 5, 71, -57]
>>> xs.extend([7,8,9])      # Adds several slots to the end.
>>> xs
[13, 5, 71, -57, 7, 8, 9]
>>> xs.insert(2,100)        # Adds a slot in the middle.
>>> xs
[13, 5, 100, 71, -57, 7, 8, 9]
```

MODIFYING LISTS: REMOVING

- We can remove slots from a list object:

```
>>> xs
[13, 5, 100, 71, -57, 7, 8, 9]
>>> xs.pop()     # Remove the last slot; return its value.
9
>>> xs
[13, 5, 100, 71, -57, 7, 8]
>>> xs[2]
100
>>> del xs[2]    # Remove a slot at a certain index.
>>> xs
[13, 5, 71, -57, 7, 8]
>>> xs[2]        # The other items shift left.
71
```

EXAMPLE LIST FUNCTION

- This function builds a list of integers:

```
def count_up(n):
    i = 1
    counts = []
    while i <= n:
        counts.append(i)
        i = i + 1
    return counts

>>> count_up(7)
[1, 2, 3, 4, 5, 6, 7]
```

EXAMPLE LIST FUNCTION

- This function builds a number's divisor sequence:

```
def divisor_list(number):
    sequence = [1]
    divisor = 2
    while divisor < number:
        if number % divisor == 0:
            sequence.append(divisor)
        sequence.append(number)
        return sequence

>>> divisor_list(35)
[1, 5, 7, 35]
>>> divisor_list(1)
[1]
>>> divisor_list(7)
[1, 7]
>>> divisor_list(36)
[1, 2, 3, 4, 6, 9, 12, 18, 26]
```

EXAMPLE LIST PROCEDURE

- This function modifies a list.

```
def rotate_right(xs):
    if len(xs) > 1:
        last = xs.pop()
        xs.insert(0,last)
```

- Calling `rotate_right` has the ***side effect*** of changing the list you give it:

```
>>> dsForSixteen = divisors_list(16)
>>> dsForSixteen
[1, 2, 4, 8, 16]
>>> rotate_right(csForSix)
>>> csForSix
[16, 1, 2, 4, 8]
>>> rotate_right(csForSix)
>>> csForSix
[8, 16, 1, 2, 4]
```

PYTHON LIST SUMMARY

- List creation via enumeration, concatenation, repetition, slicing:

```
[3,1,7]   []   [1,2]+[3,4,5]
```

- Accessing contents by index; list length:

```
xs[3]   xs[-1]   len(xs)
```

- Updating contents by indexed assignment:

```
xs[3] = 5
```

- Modifying/mutating a list object:

```
xs.append(5)    xs.extend([8,9,10])    xs.insert(2,357)
xs.pop()        del xs[6]
```

- Checking membership, content equality, object identity:

```
3 in xs      xs == [1,2,3]     xs is ys
```

- Scan according to index using a **`while`** loop:

```
i = 0
while i < len(xs):
    print(xs[i])
    i = i + 1
```

LIST "ARITHMETIC"

- We can build new lists from other list's contents using `+` and `*`:

```
>>> [1,2,17] + [111,8]
[1, 2, 17, 111, 8]
>>> [1,2,17] * 4
[1, 2, 17, 1, 2, 17, 1, 2, 17, 1, 2, 17]
>>> [1,2,17] + []
[1, 2, 17]
>>> [] + [1,2,17]
[1, 2, 17]
>>> [1,2,17] * 1
[1, 2, 17]
>>> [1,2,17] * 0
[]
>>> [] * 4
[]
>>> [] + []
[]
```

LIST "SLICING"

- We can build new lists by copying portions of other lists:

```
>>> xs = [45,1,8,17,100,6]
>>> xs
[45, 1, 8, 17, 100, 6]
>>> xs[2:5]              # Build a new list from the 2,3,4 slice.
[8, 17, 100]
>>> xs[2:4]              # Build a new list from the 2,3 slice.
[8, 17]
>>> xs[:4]               # Build a new list from the 0,1,2,3 slice.
[45, 1, 8, 17]
>>> xs[4:]               # Build a new list from the 4,5 slice.
[100, 6]
>>> ys = xs[:]           # Build a new list as a full copy.
>>> xs[1] = 121
>>> xs
[45, 121, 8, 17, 100, 6]
>>> ys
[45, 1, 8, 17, 100, 6]
```

LISTS OF LISTS

- Lists can be stored within other lists.

```
>>> lls = [[45,19],[8],[17,100,6],[]]
>>> lls[2]
[17, 100, 6]
>>> lls[2][0]
17
>>> lls[2][0] = 7777
>>> lls
[[45, 19] ,[8] ,[7777, 100, 6], []]
>>> lls[0].pop()
19
>>> lls[0].extend([0,0,0])
>>> lls
[[45,19,0,0,0],[8],[7777,100,6],[]]
>>> lls.append([5,4,3,2])
>>> lls
[[45, 19, 0, 0, 0], [8], [7777, 100, 6], [], [5, 4, 3, 2]]
```

PYTHON LIST SUMMARY ENHANCED

- List creation via enumeration, concatenation, repetition, slicing:

```
[3,1,7]  []  [1,2]+[3,4,5]  [1,2]*4  xs[3:5]  xs[3:]  xs[:]
```

- Accessing contents by index; list length:

```
xs[3]  xs[-1]  len(xs)
```

- Updating contents by indexed assignment:

```
xs[3] = 5
```

- Modifying/mutating a list object:

```
xs.append(5)    xs.extend([8,9,10])    xs.insert(2,357)
xs.pop()        del xs[6]
```

- Checking membership, content equality, object identity:

```
3 in xs      xs == [1,2,3]     xs is ys
```

- Scan according to index using a **`while`** loop:

```
i = 0
while i < len(xs):
    print(xs[i])
    i = i + 1
```

TWO PRINTING PROCEDURES

- This procedure outputs the contents of a list.

```
def output_using_while(xs):
    i = 0
    while i < len(xs):
        print(xs[i])
        i = i + 1
```

- This procedure also outputs the contents of a list.

```
def output_using_for(xs):
    for x in xs:
        print(x)
```

PYTHON LIST SUMMARY ENHANCED WITH FOR

- List creation via enumeration, concatenation, repetition, slicing:

  `[3,1,7]  []  [1,2]+[3,4,5]  [1,2]*4  xs[3:5]  xs[3:]  xs[:]`
- Accessing contents by index; list length:

  `xs[3]  xs[-1]  len(xs)`
- Updating contents by indexed assignment:

  `xs[3] = 5`
- Modifying/mutating a list object:

  `xs.append(5)` `xs.extend([8,9,10])` `xs.insert(2,357)`
  `xs.pop()` `del xs[6]`
- Checking membership, content equality, object identity:

  `3 in xs` `xs == [1,2,3]` `xs is ys`
- Scan according to index using a **`while`** loop.
- Loop through the contents using a **`for`** loop.

OUR SECOND DATA STRUCTURE: PYTHON DICTIONARIES

- Python lets you store a collection of *associations*

```
>>> d = {"bob":35, "mel":24, "betty":29}
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> d['bob']
35
>>> d['mel']
24
```

- This is a built-in data structure called a Python ***dictionary***.
  - A dictionary contains a collection of ***entries***.
  - The left part of each entry is called its ***key***.
  - The right part is that key's ***associated value***.
  - There is *at most one entry* for a key.
- A Python dictionary is our 2nd explicit example of a Python (data) ***object***

OUR SECOND DATA STRUCTURE: PYTHON DICTIONARIES

- Python lets you store a collection of *associations*

```
>>> d = {"bob":35, "mel":24, "betty":29}
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> d['bob']
35
>>> d['mel']
24
```

- This is a built-in data structure called a Python ***dictionary***.
  - ➡ It's also called a "key-value mapping", or sometimes just a "map".
  - ➡ Sometimes it's called a "hash table" or just "hashmap"
  - In some languages, you mimic a dictionary with an "association list:"

```
d = [[“bob”, 35], [“mel”,24], [“betty”,29]]
```

MODIFYING A DICTIONARY'S CONTENTS

- A Python dictionary is also a ***mutable*** data structure.
  - ➡ You can add new key-value pairs, or modify the associated value to a key.
  - ➡ The syntax for adding a new entry and updating an existing entry is the same

```
>>> d = {"bob":35, "mel":24, "betty":29}
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> d['mel']
24
>>> d['mel'] = 25
>>> d['mel']
25
>>> d
{'bob': 35, 'mel': 25, 'betty': 29}
>>> d['lou'] = 87
>>> d
{'bob': 35, 'mel': 24, 'betty': 29, 'lou': 87}
```

DICTIONARY CONTENT CHECKS

```
>>> d = {"bob":35, "mel":24, "betty":29, "lou": 87}
>>> 'mel' in d      # Does the dictionary contain a key?
True
>>> 'jim' in d
False
>>> 35 in d
False
>>> e =  {"lou": 87,"mel":24, "betty":29, "bob":35}
>>> e == d          # Are the dictionary's contents the same?
True
>>> e is d          # Are they the same object?
False
>>> len(d)          # Get the number of entries.
4
```

BUILDING AND MODIFYING A DICTIONARY

```
>>> d = {}
>>> d['bob'] = 35
>>> d['betty'] = 29
>>> d['mel'] = 24
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> del d['betty']
>>> d
{'bob': 35, 'mel': 24}
```

LOOPING

```
>>> d = {}
>>> d = {“bob”:35, “betty”:29, “mel”:24}
>>> for k in d:
...     print(k + ” -> ” + str(d[k]))
...
bob -> 35
betty -> 29
mel -> 24
>>>
```

- A `for` loop runs through the ***keys*** of the dictionary.
  - ➡ You can then look up the associated value.

PYTHON DICTIONARY SUMMARY

- List creation via enumeration of some associations:

```
{'a':89,'b':4}      {}
```

- Accessing contents by key; dictionary size:

```
d['a']                  len(d)
```

- Updating an entry's associated value with key re-assignment:

```
d['a'] = 88
```

- Modifying/mutating a dictionary to add/remove entries:

```
d['c'] = 111
del d['b']
```

- Checking key inclusion, content equality, object identity:

```
'a' in d        d == {'e':78}       d1 is d2
```

- Loop through the keys using a `for` loop.

LIST "ARITHMETIC"

- We can build new lists from other list's contents using `+` and `*`:

```
>>> [1,2,17] + [111,8]
[1, 2, 17, 111, 8]
>>> [1,2,17] * 4
[1, 2, 17, 1, 2, 17, 1, 2, 17, 1, 2, 17]
>>> [1,2,17] + []
[1, 2, 17]
>>> [] + [1,2,17]
[1, 2, 17]
>>> [1,2,17] * 1
[1, 2, 17]
>>> [1,2,17] * 0
[]
>>> [] * 4
[]
>>> [] + []
[]
```

LIST "SLICING"

- We can build new lists by copying portions of other lists:

```
>>> xs = [45,1,8,17,100,6]
>>> xs
[45, 1, 8, 17, 100, 6]
>>> xs[2:5]              # Build a new list from the 2,3,4 slice.
[8, 17, 100]
>>> xs[2:4]              # Build a new list from the 2,3 slice.
[8, 17]
>>> xs[:4]               # Build a new list from the 0,1,2,3 slice.
[45, 1, 8, 17]
>>> xs[4:]               # Build a new list from the 4,5 slice.
[100, 6]
>>> ys = xs[:]           # Build a new list as a full copy.
>>> xs[1] = 121
>>> xs
[45, 121, 8, 17, 100, 6]
>>> ys
[45, 1, 8, 17, 100, 6]
```

LISTS OF LISTS

- Lists can be stored within other lists.

```
>>> lls = [[45,19],[8],[17,100,6],[]]
>>> lls[2]
[17, 100, 6]
>>> lls[2][0]
17
>>> lls[2][0] = 7777
>>> lls
[[45, 19] ,[8] ,[7777, 100, 6], []]
>>> lls[0].pop()
19
>>> lls[0].extend([0,0,0])
>>> lls
[[45,19,0,0,0],[8],[7777,100,6],[]]
>>> lls.append([5,4,3,2])
>>> lls
[[45, 19, 0, 0, 0], [8], [7777, 100, 6], [], [5, 4, 3, 2]]
```

PYTHON LIST SUMMARY

- List creation via enumeration, concatenation, repetition, slicing:

```
[3,1,7]  []  [1,2]+[3,4,5]  [1,2]*4  xs[3:5]  xs[3:]  xs[:]
```

- Accessing contents by index; list length:

```
xs[3]  xs[-1]  len(xs)
```

- Updating contents by indexed assignment:

```
xs[3] = 5
```

- Modifying/mutating a list object:

```
xs.append(5)    xs.extend([8,9,10])    xs.insert(2,357)
xs.pop()        del xs[6]
```

- Checking membership, content equality, object identity:

```
3 in xs      xs == [1,2,3]     xs is ys
```

- Scan according to index using a **`while`** loop:

```
i = 0
while i < len(xs):
    print(xs[i])
    i = i + 1
```

TWO PRINTING PROCEDURES

- This procedure outputs the contents of a list.

```
def output_using_while(xs):
    i = 0
    while i < len(xs):
        print(xs[i])
        i = i + 1
```

- This procedure also outputs the contents of a list.

```
def output_using_for(xs):
    for x in xs:
        print(x)
```

PYTHON LIST SUMMARY

- List creation via enumeration, concatenation, repetition, slicing:

  `[3,1,7]  []  [1,2]+[3,4,5]  [1,2]*4  xs[3:5]  xs[3:]  xs[:]`

- Accessing contents by index; list length:

  `xs[3]  xs[-1]  len(xs)`

- Updating contents by indexed assignment:

  `xs[3] = 5`

- Modifying/mutating a list object:

  `xs.append(5)` `xs.extend([8,9,10])` `xs.insert(2,357)`
  `xs.pop()` `del xs[6]`

- Checking membership, content equality, object identity:

  `3 in xs` `xs == [1,2,3]` `xs is ys`

- Scan according to index using a **`while`** loop.
- Loop through the contents using a **`for`** loop.

OUR SECOND DATA STRUCTURE: PYTHON DICTIONARIES

- Python lets you store a collection of *associations*

```
>>> d = {"bob":35, "mel":24, "betty":29}
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> d['bob']
35
>>> d['mel']
24
```

- This is a built-in data structure called a Python ***dictionary***.
  - A dictionary contains a collection of ***entries***.
  - The left part of each entry is called its ***key***.
  - The right part is that key's ***associated value***.
  - There is *at most one entry* for a key.
- A Python dictionary is our 2nd explicit example of a Python (data) ***object***

OUR SECOND DATA STRUCTURE: PYTHON DICTIONARIES

- Python lets you store a collection of *associations*

```
>>> d = {"bob":35, "mel":24, "betty":29}
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> d['bob']
35
>>> d['mel']
24
```

- This is a built-in data structure called a Python ***dictionary***.
  - ➡ It's also called a "key-value mapping", or sometimes just a "map".
  - ➡ Sometimes it's called a "hash table" or just "hashmap"
  - In some languages, you mimic a dictionary with an "association list:"

```
d = [[“bob”, 35], [“mel”,24], [“betty”,29]]
```

MODIFYING A DICTIONARY'S CONTENTS

- A Python dictionary is also a ***mutable*** data structure.
  - ➡ You can add new key-value pairs, or modify the associated value to a key.
  - ➡ The syntax for adding a new entry and updating an existing entry is the same

```
>>> d = {"bob":35, "mel":24, "betty":29}
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> d['mel']
24
>>> d['mel'] = 25
>>> d['mel']
25
>>> d
{'bob': 35, 'mel': 25, 'betty': 29}
>>> d['lou'] = 87
>>> d
{'bob': 35, 'mel': 24, 'betty': 29, 'lou': 87}
```

DICTIONARY CONTENT CHECKS

```
>>> d = {"bob":35, "mel":24, "betty":29, "lou": 87}
>>> 'mel' in d      # Does the dictionary contain a key?
True
>>> 'jim' in d
False
>>> 35 in d
False
>>> e =  {"lou": 87,"mel":24, "betty":29, "bob":35}
>>> e == d          # Are the dictionary's contents the same?
True
>>> e is d          # Are they the same object?
False
>>> len(d)          # Get the number of entries.
4
```

BUILDING AND MODIFYING A DICTIONARY

```
>>> d = {}
>>> d['bob'] = 35
>>> d['betty'] = 29
>>> d['mel'] = 24
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> del d['betty']
>>> d
{'bob': 35, 'mel': 24}
```

LOOPING

```
>>> d = {}
>>> d = {“bob”:35, “betty”:29, “mel”:24}
>>> for k in d:
...     print(k + ” -> ” + str(d[k]))
...
bob -> 35
betty -> 29
mel -> 24
>>>
```

- A `for` loop runs through the ***keys*** of the dictionary.
  - ➡ You can then look up the associated value.

PYTHON DICTIONARY SUMMARY

- List creation via enumeration of some associations:

```
{'a':89,'b':4}      {}
```

- Accessing contents by key; dictionary size:

```
d['a']                  len(d)
```

- Updating an entry's associated value with key re-assignment:

```
d['a'] = 88
```

- Modifying/mutating a dictionary to add/remove entries:

```
d['c'] = 111
del d['b']
```

- Checking key inclusion, content equality, object identity:

```
'a' in d        d == {'e':78}       d1 is d2
```

- Loop through the keys using a `for` loop.